Title	OSS に対するコンポーネントの整合性を考慮した最適バージョンアップ時期の推定に関する一考察(不確実性を含む意思決定の数理とその応用)
Author(s)	田村, 慶信; 山田, 茂
Citation	数理解析研究所講究録 (2007), 1548: 194-201
Issue Date	2007-04
URL	http://hdl.handle.net/2433/80818
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
第1548巻 2007年 194-201

OSSに対するコンポーネントの整合性を考慮した最適バージョンアップ時期の推定に関する一考察

田村 慶信

広島工業大学情報学部
情報工学科
〒731-5193
広島市佐伯区三宅 2-1-1
Phone: 082-921-6042
Fax: 082-921-8978
E-mail: tam@cc.it-hiroshima.ac.jp

山田 茂

鳥取大学工学部
社会開発システム工学科
〒680-8552
鳥取市湖山町南 4-101
Phone: 0857-31-5303
Fax: 0857-31-0882
E-mail: yamada@sse.tottori-u.ac.jp

概要

オープンソースプロジェクトの下で開発されたオープンソースソフトウェアは，世界中の誰もが開発に参加でき，ソースコードが公開され，誰でも自由に改変することが可能なソフトウェアであることから，この数年急激に普及してきている．特に，オープンソースソフトウェアのリリース候補版において十分な信頼性を確認することは，正式版リリース後のユーザに対する信頼や人気に大きくかかわるだけでなく，オープンソースソフトウェアの保守コストや保守労力の増大に深く関係する．したがって，リリース候補版の信頼性を評価するとともに，最適なバージョンアップ時期を推定することはオープンソースソフトウェア開発において重要な段階となる．本論文では，最適なバージョンアップ時期の推定方法について提案する．

1 はじめに

現在，分散ソフトウェア共同開発は，同一企業内における開発形態から，複数のソフトウェアハウスや同一企業内，複数の企業間での遠隔地間共同開発，さらには，多くの開発者が協調しながら開発を行うオープンソースプロジェクトなどの様々な形態が存在する．これに伴い，最近のソフトウェア開発は，クライアント/サーバ・システムやWebプログラミング，オブジェクト指向開発，ネットワーク環境での分散開発といった新しい開発技術が多用されるようになってきている [1]．特に，ネットワーク環境を利用して開発されたオープンソースソフトウェア（Open Source Software，以下OSSと略す）は，世界中の誰もが開発に参加でき，ソースコードが公開され，誰でも自由に改変することが可能なソフトウェアであることから，組込みシステムやサーバ用途として広く採用されている．また，OSSの代表格ともいうべきLinux[1]は市場でサーバ用途として有効であると認められ，この数年急激に普及してきている [2]．

オープンソースプロジェクトは，世界中に分散した複数のコンポーネントが，何らかのプラットフォーム上で的確に機能しているとすれば，開発が迅速であるばかりか，競争原理によりある評価基準の下でベストなものが残っていくと考えられる．オープンソースプロジェクトが分散型開発モデルを採用して成功した例として，GNU/Linuxオペレーティングシステムや Apache ウェブサーバなど[2]が挙げられる [3]．

一方，OSSの利用に関しては，未だに多くの不安が残されている．まず第1に，システム導入後のサポートおよび品質上の問題といった利用者側の一般的な不安である．第2に，OSSは本当にビジネスになるのか，オープンソースのソフトウェアを事業化することによって自社製のソフトウェア商品までが市場を失うことにならないか，といった開発者側の不安である [4]．特に，サポートや品質上の問題については，OSSの普及を妨げる大きな要因として考えられている．また，OSSに対する現在の研究動向としては，設計工程や開発手法を対象とした文献はいくつか提案されているが [5, 6]，動的解析に基づいた信頼性評価に関する研究はほとんど行われていないのが現状である．

本論文では，こうしたオープンソースプロジェクトの下で開発されているOSSに対して，テスト管理に関する問題の1つとして信頼性評価法について議論するとともに，実際に公開されているOSSに対する信頼性評価法の適用可能性と，その有効性について考察する．特に，システム全体を構成する複数のコンポーネント，いわゆる各

[1] Linux は，Linus Torvalds の米国およびその他の国における登録商標あるいは商標である．
[2] その他記載している会社名，商品名は一般に各社の商標または登録商標である．

ソフトウェアコンポーネントの重要度を推定するためにニューラルネットワークを適用する．これにより，バグトラッキングシステム上から得られたデータに基づき，各コンポーネント間の相互作用を包括した信頼性評価法として利用できるものと考える．同時に，ソフトウェア信頼度成長モデル（software reliability growth model，以下SRGMと略す）[7]に基づき各コンポーネント間の相互作用を包括した信頼性評価法を提案する．また，実際のフォールト発見数データに対する数値例を示す．さらに，OSSの最適なバージョンアップ時期の推定法について考察する．

2 各コンポーネントに対する信頼性評価

ソフトウェアの信頼性評価手法の開発において，各コンポーネントでのデバッグの状況やその良し悪しが，システム全体の信頼性に与える影響を考慮しようとする場合，プログラムパス，コンポーネントの規模，フォールト報告者のスキルなどの，様々に絡み合った要因を捉える必要があると考えられる．こうした複雑な状況下でシステム全体の信頼性に対する各コンポーネントの影響度合いを推定するために，本論文ではニューラルネットワークを適用する．これにより，バグトラッキングシステム上から採取されたデータのみに基づいて機械的に各コンポーネントのシステム全体に与える重要度を推定することが可能となる．

OSSにおいて，システム全体の信頼性に対する各コンポーネントの影響を考えた場合，コンポーネントの規模，フォールト報告者のスキル，フォールト修正の状態，コンポーネントの開発時間，コンポーネント間のパスの数，コンポーネント間の入出力データ量といった様々な要因を考慮する必要がある．こうした複雑な状態を適当な仮定の下で物理的な意味からモデル化することは困難である．したがって，本論文では，各コンポーネント間の内部状態をブラックボックスとして捉えるためにニューラルネットワーク[8]を適用する．すなわち，入力と出力の関係から，その内部構造をニューラルネットワークにより学習させることによって，各コンポーネントがシステム全体の信頼性に与える影響度合いを推定する．これにより，バグトラッキングシステム上から採取されたデータのみに基づいた信頼性評価が可能となることから，実利用上においても容易に適用できるものと考える．本論文においては簡単のために3層ニューラルネットワークを適用する．

まず，$w_{ij}^1(i=1,2,\cdots,I;j=1,2,\cdots,J)$を入力層と中間層の結合係数，また$w_{jk}^2(j=1,2,\cdots,J;k=1,2,\cdots,K)$は中間層と出力層の結合係数とする．さらに，$x_i(i=1,2,\cdots,I)$は正規化された入力データを表し，本論文では，フォールト報告者により致命的であると判断されたフォールト数，特定のOSにおいて発見されたフォールト数，システムの内部構造に習熟した修正者のフォールト修正数，システムの内部構造に習熟した発見者のフォールト発見数とした．ここで，入力層，中間層，出力層におけるユニットの数を，各々I個，J個，およびK個とする．また，各々の層のユニットを示すインデックスをi，j，およびkとする．ここで，各々の層のユニットの出力をh_j, y_kとすると，

$$h_j = f\left(\sum_{i=1}^{I} w_{ij}^1 x_i\right), \tag{1}$$

$$y_k = f\left(\sum_{j=1}^{J} w_{jk}^2 h_j\right), \tag{2}$$

となる．但し，$f(\cdot)$はシグモイド型関数であり，

$$f(x) = \frac{1}{1+e^{-\theta x}}, \tag{3}$$

として表される．ここで，θはゲインと呼ばれる定数である．ネットワークの学習を行うために，誤差逆伝播法を用いる．ニューラルネットワークの出力層における値を$y_k(k=1,2,\cdots,K)$とし，教師パターンを$d_k(k=1,2,\cdots,K)$とすると，式(2)のy_kの評価は次式で与えられる．

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2. \tag{4}$$

ここで，教師パターン$d_k(k=1,2,\cdots,K)$には，各コンポーネントにおける累積発見フォールト数データの正規化された値を採用する．すなわち，各コンポーネントにおける累積フォールト発見数データに基づいて，各コンポーネントの重み係数とそれに影響を及ぼす要因の結合状態の特徴をニューラルネットワークの結合係数に蓄積させ，ある時点における各コンポーネントの重要度の推定・予測が可能なモデルを考える．式(4)の条件のもと

に，各結合係数が最急降下法にて以下のように決定される．

$$w_{jk}^2(\sigma+1) = w_{jk}^2(\sigma)+\epsilon(y_k-d_k)\cdot f'\left(\sum_{j=1}^{J} w_{jk}^2(\sigma)h_j\right)h_j, \tag{5}$$

$$w_{ij}^1(\sigma+1) = w_{ij}^1(\sigma)+\epsilon\sum_{k=1}^{K}(y_k-d_k)\cdot f'\left(\sum_{j=1}^{J} w_{jk}^2(\sigma)h_j\right)\cdot w_{ij}^1(\sigma)f'\left(\sum_{i=1}^{I} w_{ij}^1(\sigma)x_i\right)x_i. \tag{6}$$

ここで，σ は更新のサイクル，ϵ は学習の係数を表す．式 (5) および式 (6) の更新により求められた w_{ij}^1 および w_{jk}^2 から，式 (1) および式 (2) により，ニューラルネットワークの出力層における値 $y_k(k=1,2,\cdots,K)$ を算出することができる．これにより，各コンポーネントに対する重みパラメータ $p_i(i=1,2,\cdots,K)$ を次の式により導出できる．

$$p_i = \frac{y_i}{\sum_{i=1}^{K} y_i}. \tag{7}$$

本論文では，ニューラルネットワークにおいて推定された各コンポーネントの重要度 p_i を，システム全体の信頼性に対する各コンポーネントの影響度合いと定義する．

3 システム全体に対する信頼性評価

3.1 一般化対数型ポアソン実行時間モデル

本論文で取り上げる OSS の動作環境は，様々なアプリケーションソフトウェアから影響を受け易く，従来のような同一組織内で開発され，単体で動作するソフトウェアシステムとは大きく環境が異なる．こうしたソフトウェア間の相互作用により，発見されるフォールト数も一定の値に収束することなく，将来的には増加し続けるものと考えられる．

本論文では，検出可能フォールト数が無限であると仮定された NHPP に基づく対数型ポアソン実行時間モデルを適用する．時間区間 $(0,t]$ で発見される総期待フォールト数を表す平均値関数 $\mu(t)$ は，

$$\mu(t) = \frac{1}{\theta-P}\ln[\lambda_0(\theta-P)t+1] \quad (0<\theta,\ 0<\lambda_0,\ 0<P<1), \tag{8}$$

により与えられる．ここで，パラメータ λ_0 は初期故障強度，パラメータ θ はソフトウェア故障 1 個当りの故障強度の減少率を表す．また，パラメータ P はシステム全体に及ぼすコンポーネントの影響率を表す．これは，各コンポーネントに対してニューラルネットワークを用いて推定されたパラメータ y_i の平均値により表されるものと定義する [9, 10, 11].

3.2 ソフトウェア信頼性評価尺度

式 (8) の平均値関数をもつ NHPP モデルから，種々のソフトウェア信頼性評価のための定量的尺度を導出できる．

瞬間フォールト発見率は強度関数により表すことができる．これは，単位時間当りに発見されるフォールト数として定義される．瞬間フォールト発見率は，式 (8) から以下のように導出できる．

$$\mu_d(t) = \frac{d\mu(t)}{dt}. \tag{9}$$

平均ソフトウェア故障発生時間間隔（mean time between software failures: MTBF）は，ソフトウェア 故障の発生頻度を表すのに有益な尺度である．また，MTBF が大きな値を取ることは，それだけフォールトが発見し難くなり，ソフトウェア信頼性が向上したと判断できることになる．任意の時刻 t における瞬間 MTBF（instantaneous MTBF: $\mathrm{MTBF_I}$）および累積 MTBF（cumulative MTBF: $\mathrm{MTBF_C}$）は，以下のように導出できる．

任意の時刻 t における瞬間的なフォールト発見間隔の平均を意味する瞬間 MTBF は，

$$MTBF_I(t) = \frac{1}{d\mu(t)/dt}, \tag{10}$$

表 1： Thunderbird の各コンポーネントに対する要因別データ.

Component Name	Number of Fault Level (Critical)	OS (Windows)	Assigned to (mscott)	Reporter (floeff)	Total Number of Faults
Account Manager	9	88	127	1	130
Address Book	4	59	102	3	105
Build Config	1	3	14	0	15
General	72	293	538	3	558
Help Documentation	0	1	3	0	5
Installer	3	28	33	1	33
Mail Window Front End	58	578	982	6	1013
Message Compose Window	15	127	227	6	233
Migration	3	36	51	0	51
Preferences	2	37	92	5	96
RSS	3	45	76	0	78
Total	170	1295	2245	25	2317

表 2： Thunderbird の各コンポーネントに対する重み係数.

Component Name	Weight parameter
Account Manager	0.0537
Address Book	0.0487
Build Config	0.0091
General	0.2377
Help Documentation	0.0054
Installer	0.0138
Mail Window Front End	0.4347
Message Compose Window	0.0951
Migration	0.0266
Preferences	0.0425
RSS	0.0327

となる.

運用開始時点から考えたときの発見フォールト 1 個当りに要する発見時間の平均を意味する累積 MTBF は,

$$MTBF_C(t) = \frac{t}{\mu(t)}, \tag{11}$$

により表すことができる.

4 実測データに適用した数値例

4.1 各コンポーネントに対する信頼性評価結果

実際のオープンソースプロジェクトにおけるバグトラッキングシステムから採取されたフォールトデータを適用した数値例を示す. 本論文で提案された信頼性評価法の性能評価を行うために, Mozilla.org プロジェクト[3]で開発が進められているメーラの Thunderbird[4][12] と呼ばれる OSS を取り上げる.

各コンポーネントに関して, システム全体の信頼性に影響を及ぼすと考えられる要因別データ一覧を表 1 に示す. ニューラルネットワークを適用するにあたり, 表 1 におけるデータを入力データとした. 2 のニューラルネットワークに基づく各 OSS の各コンポーネントに対する重みパラメータ $p_i(i = 1, 2, \cdots, n)$ の推定結果を表 2 に示す. 表 2 から, Mail Window Front End コンポーネントに対する重要度が最大であることが分かる. 一方, Help Documentation コンポーネントに対する重要度は最小であることが確認できる.

[3]Mozilla と Mozilla のロゴは Mozilla Foundation の登録商標である.
[4]Thunderbird と Thunderbird のロゴは Mozilla Foundation の米国及びその他の国における商標または登録商標である.

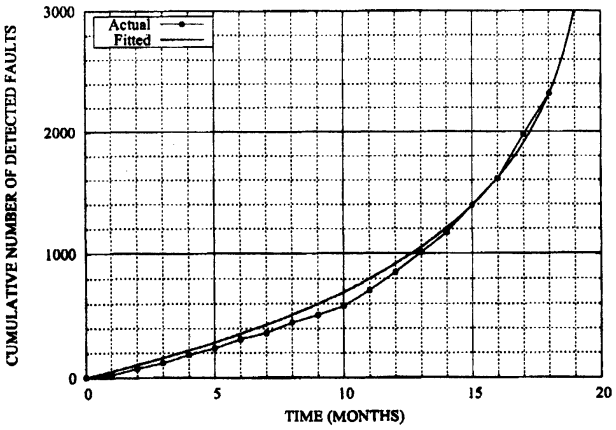

図 1： 推定された累積フォールト発見数の期待値，$\widehat{\mu}(t)$.

4.2 システム全体に対する信頼性評価結果

次に，システム全体に対する信頼性評価結果の一例を示す．まず，ニューラルネットワークおよび最尤法を用いて推定された各パラメータの推定値は，

$$\widehat{\theta} = 0.20660,\ \widehat{\lambda_0} = 50.185,\ \widehat{P} = 0.20759,$$

となる．式 (8) における累積フォールト発見数の期待値の推定値 $\widehat{\mu}(t)$ を図 1 に示す．

5 最適バージョンアップ

5.1 最適バージョンアップ時期の推定

OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定することは，リリース後の信頼性維持や進捗度管理に役立つと考えられる．本論文では，オープンソースプロジェクトの下で開発された Thunderbird[12] を一例に挙げ，OSS のバージョンアップ時期の推定方法について議論する．

バージョンアップには大きく分けてマイナーバージョンアップとメジャーバージョンアップがある．しかしながら，どの程度の改訂で区別されるという明確な基準がある訳ではない．マイナーバージョンアップは，旧版にいくつかの細かい機能が追加されたり，性能が若干向上した場合に実施される．ただし，機能の不具合やセキュリティ上の脆弱性を修復するような，クリティカルなフォールトがいくつか発見され緊急に修正を施す必要がある場合に実施される改訂はマイナーバージョンアップではなく，バグフィックスと呼ばれている．一方，メジャーバージョンアップは，OSS 自体の機能が大きく変更されたり，大型の新機能の追加や，性能が劇的に向上した場合に実施される．OSS におけるマイナーバージョンアップは，不定期かつ頻繁に実施されていることから，その推定時期を予測することは困難であり，たとえマイナーバージョンアップ時期が推定できたとしても，その有益性は小さいと考えられる．したがって，本論文では，メジャーバージョンアップを対象とし，前回のバージョンアップ期間に基づいて，次期バージョンアップ時期を予測するための手法について考察する．

5.2 総開発労力の定式化

OSS の開発に伴う総開発労力を定式化し，総開発労力を最小にする時刻を最適バージョンアップ時刻と定義することにより，バージョンアップ後の信頼性維持や進捗度管理に役立つものと考える．まず，総開発労力を定式化するために，以下のパラメータを定義する．

m_1: フォールト修正に伴う修正労力 $(m_1 > 0)$,

m_2: 単位時間当りの開発労力 $(m_2 > 0)$,

m_3: メジャーバージョンアップ後のフォールト修正に伴う保守労力 $(m_3 > m_1)$,

m_4: メジャーバージョンアップ後の単位時間当りの保守労力 $(m_4 > m_2)$.

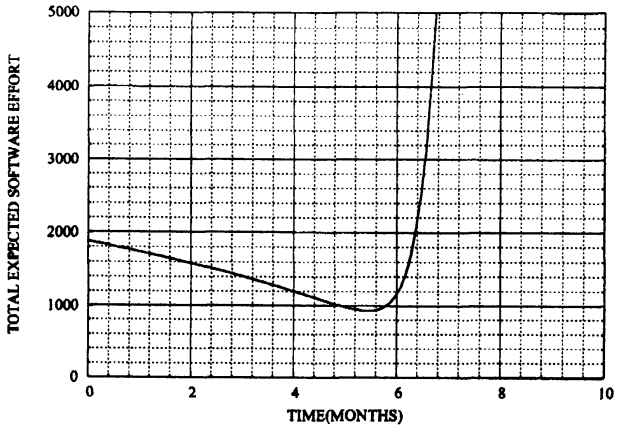

図 2： 推定された総期待開発労力.

よって，以下のようなシステム全体に対する期待開発労力が得られる.

$$E_1(t) = m_1 \cdot \mu(t) + m_2 \cdot t. \tag{12}$$

一方，メジャーバージョンアップ後の保守労力は以下のように定式化できる.

$$E_2(t) = m_3\{\mu(t_0) - \mu(t)\} + m_4 t. \tag{13}$$

ここで，t_0 は過去のバージョンアップ期間の平均値を表す.

さらに，時間区間 $(0, t_0]$ を越えた場合において，コンポーネントを新規に開発する際には，整合性を確認するためのペナルティ労力が課せられるものとする．本論文では，ペナルティ関数を以下のように定義する.

$$G(t) = c \cdot e^{k(t-t_0)}. \tag{14}$$

ここで，$c(>0)$ は，t_0 を越えて開発されたシステム全体に対する新規コンポーネントの割合，$k(>0)$ は，過去のメジャーバージョンアップ回数を表す.

したがって，総期待開発労力は，式 (12)，式 (13) および式 (14) より，

$$E(t) = E_1(t_i) + E_2(t) + G(t) \tag{15}$$

のように表すことができる．この式 (15) を最小にする時刻 t^* が，OSS の最適メジャーバージョンアップ時刻となる.

5.3 数値例

最適メジャーバージョンアップ時期推定の数値例として，Thunderbird を取り上げる．ここでは一例として，Ver. 1.6 から Ver. 1.7 への移行時期に基づき，Ver. 1.8 への次期バージョンアップ時刻を推定する．Ver. 1.6 は 2004 年 1 月 15 日にリリースされ，Ver. 1.7 は 2004 年 6 月 17 日にリリースされており，この間の期間は 111 日である．したがって，t_0 を 111 と仮定し，2004 年 6 月 17 日以降における式 (15) の推定された総期待開発労力を図 2 に示す．図 2 から，最適バージョンアップ時刻は Ver. 1.7 がリリースされてから約 5.45ヶ月目となり，そのときの総期待開発労力は 931.45 であることが確認できる.

5.4 パラメータの感度分析

式 (15) に含まれるパラメータ $c(>0)$ および $k(>0)$ に対する感度分析結果を示す.

まず，**5.3** の数値例に基づいて，パラメータ c を固定した状態で，パラメータ k を変化させた場合における感度分析結果を図 3 に示す．図 3 から，過去のメジャーバージョンアップ回数が増加するにつれて，総期待開発労力は増加することが確認できる．一方，メジャーバージョンアップを繰り返すにつれてシステム全体に対する習熟度が高くなり，最適メジャーバージョンアップ時刻は短くなることが分かる．さらに，パラメータ k を固定した

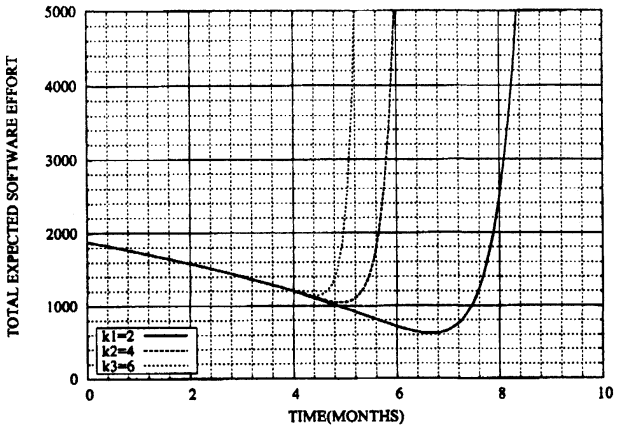

図 3： パラメータ k に対する感度分析結果.

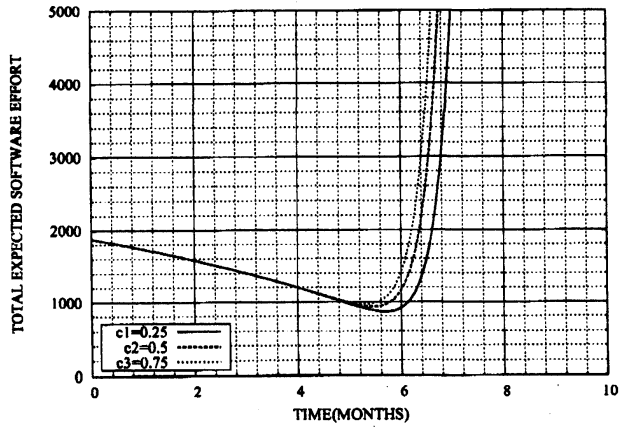

図 4： パラメータ c に対する感度分析結果.

状態で，パラメータ c を変化させた場合における感度分析結果を図 4 に示す．図 4 から，システム全体に占める新規開発コンポーネント数の割合が増加するにつれて，総開発労力が大きくなる様子が確認できる．一方，新規開発コンポーネント数の割合がリリース時期に与える影響は少ないことが分かる．

6 おわりに

本論文では，オープンソースプロジェクトの下で分散共同開発されている OSS に対する信頼性評価法について議論した．特に，ユーザ指向の信頼性評価法としてニューラルネットワークを適用することにより，各コンポーネントに対する相互作用を包括した信頼性評価法について議論した．また，実際の OSS のバグトラッキングシステムから採取されたフォールト発見数データに対する数値例を示した．

本手法により，システム全体の信頼性に与える各コンポーネントの重要度を定量的に導出することが可能となる．さらに，導出された各コンポーネントの重み係数から各コンポーネントに対する発見フォールト数の推定することが可能となることから，OSS の信頼性を定量的に把握するための手段として有効であると考える．さらに，OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定するために，最適バージョンアップ時期の推定方法について議論した．本論文において提案された手法によって，OSS のバージョンアップ後の信頼性維持や進捗度管理に役立つと考えられる．特に，最適バージョンアップ時期の推定のために，OSS の総期待開発労力を定式化するとともに，モデルに含まれるパラメータに対する感度分析を行った．しかしながら，フォールト修正に伴う修正労力や単位時間当りの開発労力に関するパラメータの決定方法が曖昧であることから，

今後はこれらのパラメータの厳密な設定方法について定義する必要がある．

オープンソースという開発スタイルは，今後も何らかの形で市場で大きな流れを作っていくものと考えられる．この流れを阻害する大きな要因として，サポートや品質上の問題が挙げられる．本論文では，こうした問題を解決するために，オープンソースプロジェクトの下で開発されたOSSに対する信頼性評価法の1例を示した．

将来的には，本手法をソフトウェアツールとして実装することにより，多くのユーザに対して容易に扱うことが可能な信頼性評価ツールとして提供していくことが重要であると考える．これまで，OSSでは信頼性を定量的に評価するという試みが行われていなかったことから，本論文において新たに提案された信頼性評価手法をOSSに適用することによって，より高品質なOSSの開発に結びつくものと考える．

謝辞

本論文の一部は，文部科学省科学研究費基盤研究 (C)（課題番号 18510124）および若手研究 (B)（課題番号 17700039）の援助を受けたことを付記する．

参考文献

[1] A. Umar, *Distributed Computing and Client-Server Systems*, Prentice Hall, New Jersey, 1993.

[2] トロン協会 ITRON 仕様検討グループ, ITRON Project Archive, http://www.sakamura-lab.org/TRON/ITRON/home-j.html

[3] E-Soft Inc., Internet Research Reports, http://www.securityspace.com/s_survey/data/index.html

[4] ソフトウェア情報センター研究会報告書, オープンソースソフトウェアの利用状況調査／導入検討ガイドラインの公表について, 東京, 2004.

[5] A. MacCormack, J. Rusnak, and C.Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Informs Journal of Management Science*, vol. 52, no. 7, pp. 1015–1030, 2006.

[6] G. Kuk, "Strategic Interaction and Knowledge Sharing in the KDE Developer Mailing List," *Informs Journal of Management Science*, vol. 52, no. 7, pp. 1031–1042, 2006.

[7] 山田 茂, ソフトウェア信頼性モデル－基礎と応用－, 日科技連出版社, 東京, 1994.

[8] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Trans. Neural Networks.*, vol. 1, pp. 239–242, 1990.

[9] 田村慶信, 山田茂, 木村光宏, "オープンソース共同開発環境に対するソフトウェア信頼性評価法に関する考察," 電子情報通信学会論文誌, vol. J88–A, no. 7, pp. 840–847, 2005.

[10] Y. Tamura and S. Yamada, "Comparison of software reliability assessment methods for open source software," *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS2005)-Volume II*, Fukuoka, Japan, July 20–22, pp. 488–492, 2005.

[11] 田村慶信, 肌附康司, 山田茂, 木村光宏, "オープンソースソフトウェアに対するユーザ指向の信頼性評価ツールの開発, " Linux Conference, 東京, 2006, http://lc.linux.or.jp/lc2006/

[12] The Mozilla Thunderbird Mail Project, Thunderbird, http://www.mozilla.org/projects/thunderbird/